D3CoS

Designing **D**ynamic **D**istributed **Co**operative Human-Machine **S**ystems

D3.05 Reference Architecture for DCoS Operation – Preliminary Version

Project Number:	269336
Classification:	Public
Work Package(s):	WP 3.4
Milestone:	M2
Document Version:	Vs. 0.1
Issue Date:	30.06.2012
Document Timescale:	Project Start Date: March 1, 2011
Start of the Document:	Month 13
Final version due:	Month 16
Keywords:	D3CoS; DCoS Software Architectures; DCoS Interfaces; communication architecture; communication infrastructure; Methods, Techniques and Tools (MTTs); software architecture requirements; architecture Vision; abstract architecture; DCoS
Compiled by:	Gerald, Temme, DLR
Authors:	Petr Benda, CTU Fabien Garcia, ENA Jean-Marc Temmos, VST Michael Nestler EAD
Technical Approval:	Catherine, Ronfle-Nadaud, ENA Andreas Lüdtke, OFF
Issue Authorisation:	Fabio Tango, CRF Andreas Lüdtke, OFF

	**D3CoS** Designing Dynamic Distributed Cooperative Human-Machine Systems	

DISTRIBUTION LIST		
Copy type[1]	Company and Location	Recipient
T	D3CoS Consortium	all D3CoS Partners

[1] Copy types: E=Email, C=Controlled copy (paper), D=electronic copy on Disk or other medium, T=Team site (Sharepoint)

	**D3CoS** Designing Dynamic Distributed Cooperative Human-Machine Systems	

RECORD OF REVISION		
Date:	Status Description:	Author:
20.03.2012	Initial version of the document	Gerald Temme
29.05.2012	Sources of requirements added	Petr Benda
13.06.2012	Chapter introduction and definitions available	Gerald Temme
20.06.2012	Existing architectures descriptions added	Gerald Temme
21.06.2012	Inputs for the architecture vision chapter available	Fabien Garcia / Jean-Marc
22.06.2012	Requirements description updated	Petr Benda
25.06.2012	Conclusions and next steps added	Gerald Temme
25.06.2012	Final version of the architecture requirement chapter available	Petr Benda
25.06.2012	Final version of the abstract architecture vision chapter available	Fabien Garcia / Jean-Marc
25.06.2012	Version for the internal review of the deliverable available	Gerald Temme
28.06.2012	Handle comments from internal reviewers	Gerald Temme
29.06.2012	Final Version available	Gerald Temme

D3CoS

Designing Dynamic Distributed Cooperative Human-Machine Systems

Table of Contents

1 Introduction

The overall objective of the D3CoS project is to develop and collect Methods, Techniques and Tools (MTTs) to support affordable development of highly innovative cooperative human-machine systems. These MTTs will enable the designers to efficiently and systematically analyse and decompose the cooperative system into its relevant subcomponents (tasks, agents, resources) and systematically specify and develop these components and the possible interaction strategies between them according to different optimisation criteria.
Important is that the focus of D3CoS is on the project phases in which a project and a DCoS are defined and specified and where prototypes are build (i.e. the concept phase, system specification phase, prototype system build).

The objective of the WP3.4 “DCoS Architectures, Interfaces and Communication Infrastructure” is to collect and extend existing DCoS software architectures as well as to develop new reusable, domain independent DCoS software architectures concepts, interfaces and communication infrastructures. They allow the operation of DCoS with modelled agents, embedded systems or other components included.

The collected existing software architectures and interfaces from the different D3CoS domains (Automotive, Manned Air Vehicle, Unmanned Air Vehicle, and Maritime) will be available as MTTs to build up and analyse DCoS already at cycle 1 (first year) for the D3CoS project.

During the next cycles 2&3 a new domain independent software architecture concept and communication interfaces will be developed. The abstract software architecture concept will be used to assess the collected DCoS software architectures. Thereby improvements for the collected software architectures will be recommended to prepare the collected DCoS software architectures for the development and presentation of the final D3CoS prototype demonstrators [D2-01].

1.1 Objective of this Document

This document describes the approach to DCoS architectures, interfaces and communication infrastructure (MTTs) improved or developed within this work package.

1.2 Structure of this Document

In chapter two of this document introduces the WP3.4 within the D3CoS project and describes the planed results of the WP3.4, as well as the WP planed process at the three D3CoS cycles.

In the following chapter three are the collected state of the art DCoS software architectures summarized.

Chapter four describes the analysed requirements from SP1 as well as additionally received requirements from the collected state of the art DCoS software architectures which were presented in chapter 3.

Chapter five introduces the abstract architecture vision, describes the influence of the on-Board and Traffic perspective on the abstract architecture as well as describes the first version of the communication architecture features.

Finally chapter six summarises the WP3.4 work within D3CoS cycle 1 and describes the next steps planned for cycle 2.

2 Definitions

This chapter show's first how the architecture work package is connected to the other D3CoS work packages. Thereafter, we will present by considering the D3CoS MTT landscape a description of the MTTs planned to be developed within this WP to fulfil the work package objectives. Finally a more detailed description of the process to develop or collect the required MTTs will be presented.

2.1 About MTTs

Before the description of expected results of the D3CoS architecture and interface work package it is important to understand the definition of an MTT – that is an instantiation of the "Methods, Techniques, and Tools" developed within D3CoS [D3-01]:

- **Methods** are rules, procedures, steps to be followed to solve problems, or to design a system (e.g. Design Patterns)
- A **Technique** is an instantiation of one method (e.g. algorithms, protocols)
- **Tools** are concrete implementations in terms of software or hardware (e.g. PED, DCoS Simulation Platforms)

2.2 About DCoS Software Architecture

Software architecture is a structured hierarchical collection of system components their descriptions and relationships. Helmut Balzert [Balzert] definition of software architecture.

DCoS software architecture describes the global characteristics of the overall DCoS components. Therefore the software architecture is a tool (in the D3CoS sense) that describes the essential components of a software platform and their interfaces that will be used to develop, simulate and evaluate a DCoS. Therefore, it is the basis for the development, for example, of DCoS Simulation Platforms. Consequently, in D3CoS the description of the software architecture is especially consumed by the Simulation Platform developers in WP3.2. These

platforms represent tools to simulate DCoS with models, humans or hardware in the loop.

2.3 About Abstract Architecture

A result of the WP3.4 will be a description of an abstract DCoS architecture. A reference for the description and meaning of abstract architectures is the
FIPA abstract architecture specification [FIPA]:

By describing systems abstractly, one can explore the relationships between fundamental elements of these agent systems. By describing the relationships between these elements, it becomes clearer how agent systems can be created so that they are interoperable. From this set of architectural elements and relations one can derive a broad set of possible concrete architectures, which will interoperate because they share a common abstract design.

2.4 Work Package within D3CoS

The DCoS architectures, interfaces and communication infrastructure work package are part of the D3CoS SP3 “Methods, Techniques & Tools”. The work package is organized by two tasks. The first task “DCoS Architecture” mainly deals with collecting, developing and accessing DCoS architectures. The second task “Interfaces to DCoS Components” focuses on the development and definition of interfaces between DCoS architecture components as well as with the needed communication infrastructure.

Figure 1 shows the architecture work package within the D3CoS project. The results of the architecture work package are consumed by the WP3.2 “DCoS Modelling and Simulation” [D3-02] to prepare the simulation platforms, by the WP3.5 “DCoS Development and Methodology” [D3-06] and by the SP2 “Demonstrators” [D2-01] to prepare the final D3CoS prototype demonstrations.

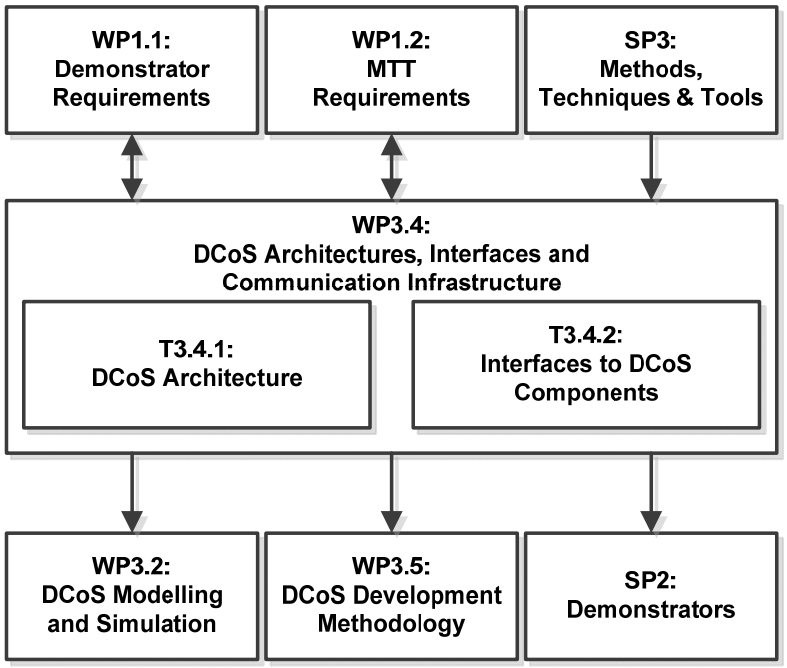


Figure 1: Work package organisation within D3CoS

The demonstrator and MTT requirements that are defined by SP1 [D1-01] [D1-02] represent the input for the architecture work package. After processing these requirements WP3.4 provides feedback to SP1 to improve the requirements. Within SP3 the main input for WP3.4 is the DCoS Generic Framework developed within WP3.1 [D3-01]. This framework provides essential definitions the main components of a DCoS, such as agents, resources, tasks, that the DCoS architecture needs to be capable of representing them.

Additionally, WP3.4 also considers the results of the other SP3 WPs, such as the Design Patterns or the DCoS XML [D3-03][D3-04] with regard to requirements that the concepts behind these developments induce for a DCoS architecture description. That is, WP3.4 does not consider specific software requirements resulting from the specific tools developed in the different WPs of SP3, that is, for example, specific interface requirements to connect a tool such as the PED to a simulation platform, such as MoSAIC [D3-02]. But it considers whether there are conceptual assumptions behind the tool, in case of PED, for example, that tasks can be described as a hierarchy of subtasks that have to be represented as

components or as characteristics of components or interactions within an abstract DCoS software architecture.

2.5 MTTs of this Work Package

Considering the D3CoS MTT Landscape (Figure 2) the abstract DCoS software architecture as developed in WP3.4 is part of the Simulation and the Implementation MTT cluster.

The MTTs of the Simulation cluster provide tools to simulate a DCoS either using models of all DCoS components, including the human agents as well as tools to simulate a DCoS with humans and, or DCoS hardware in the loop. In D3CoS this loop is used to develop, improve and evaluate solutions for the D3CoS use cases (like the cooperative lane change assistant [D2-01]).

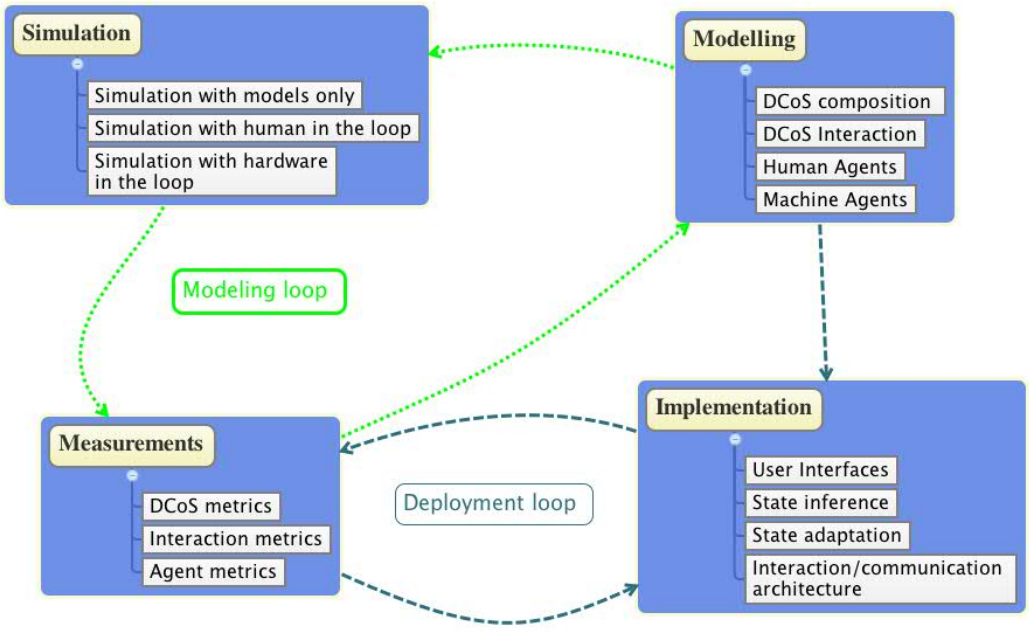


Figure 2: D3CoS MTT Landscape

We started in WP3.4 with the collection and description of already **existing DCoS software architectures** collected from the D3CoS partners (see also chapter 3). The goal was, first, to collect and describe

the state of the art of DCoS architectures as a basis from which to develop an abstract DCoS software architecture, and, second, to provide an early basis for the set-up of simulation platforms in WP3.2 that were themselves required early in the project to start and facilitate the development of the D3CoS use cases. For example, these simulation platforms were already used to perform first evaluations and experiments in the first year of the project. The collected architectures have been developed and/or applied before by the D3CoS partners and define state of the art of architectures of simulation environments to simulate distributed cooperative systems.

These existing DCoS architectures are also used as references for an **abstract DCoS architecture** (domain and architecture perspective independent) that will be developed within the D3CoS project. This abstract DCoS architecture will provide the basis for an assessment of the existing architectures to propose enhancements and allow the development of **enhanced existing DCoS architectures**.

Within the Implementation MTT cluster as part of the development loop of D3CoS (see Figure 2) WP3.4 develops new or enhances existing **Interaction / communication architectures**.

Finally the enhanced DCoS architectures of the modelling loop and the developed Interaction / communication architectures of the development loop will be used to prepare the final D3CoS demonstrators.

2.6 About Architecture MTTs

Figure 3 presents the architecture collection, definition and enhancement process and shows what results will be available at end of which D3CoS cycle (year).

During cycle 1 of the project the process concentrated on the collection and analysis of state of the art DCoS architectures from the different D3CoS domains: Automotive, Unmanned Aerial Vehicles, Manned Air Vehicles and Maritime. Some of these existing architectures were used directly to prepare simulation platforms for early DCoS experiments and evaluations. Based on the analysis of the existing architectures a first version of requirements was developed within WP1.2. Additionally,

requirements collected for the D3CoS demonstrators by WP1.1, were also taken into account as basis for the definition of an abstract DCoS architecture. This first version was used to classify the identified requirements into domain-specific and domain-independent requirements for DCoS architectures.

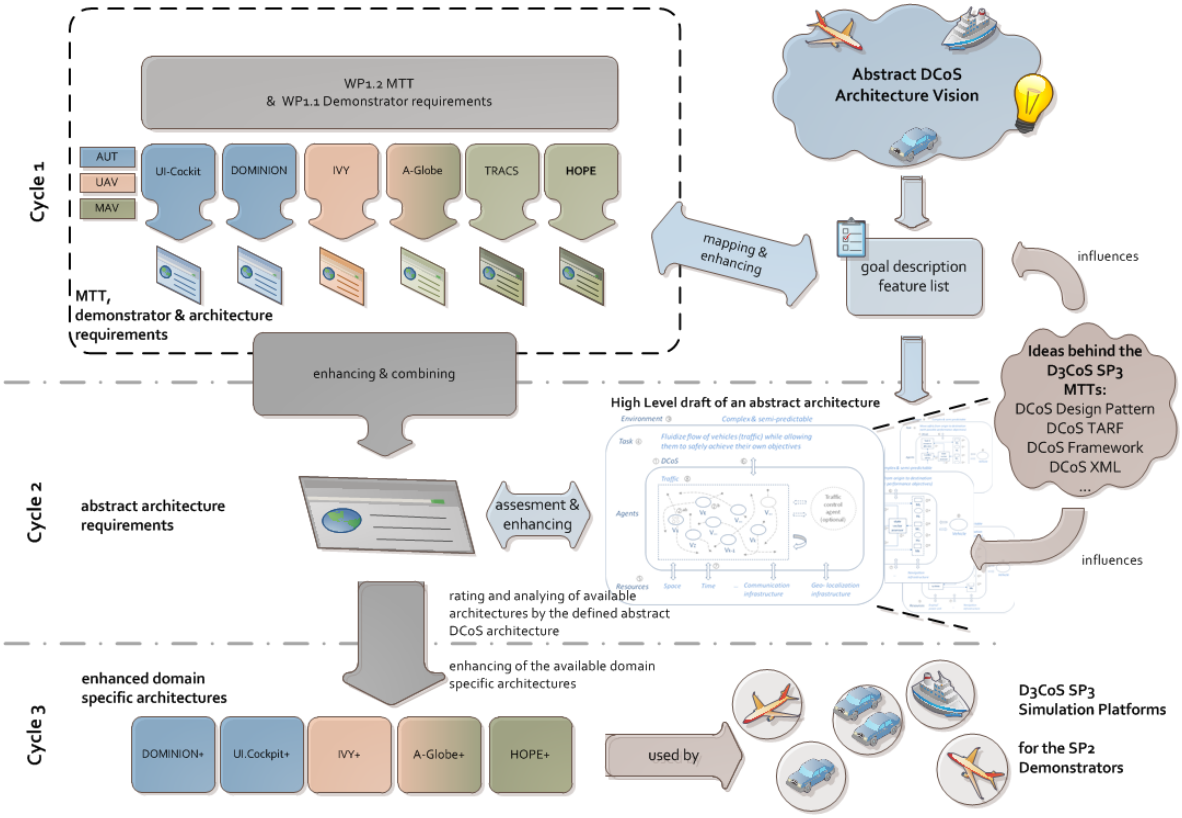


Figure 3: Work package architecture process steps

In parallel to and in continuous exchange with the collection and definition of architecture requirements the definition of an abstract DCoS architecture started. This architecture should be domain-independent as well as independent from a traffic or on-board perspective. The result of this definition process is a goal and feature description for an abstract DCoS architecture. This goal and feature description also took into account the conceptual basis underlying the different MTTs as developed in the different SP3 WPs as described above. The description provides the basis for the preparation of a high level abstract architecture definition and description (in UML) in cycle 2.

In the second D3CoS cycle the collected requirements and the defined abstract architecture goals and features are the basis for the definition and formal description (in UML) of an abstract DCoS architecture, its components, interfaces and relationships.

With the final version of the defined abstract DCoS architecture an assessment of the collected DCoS architectures will be performed to suggest improvements for the collected and existing architectures. The goal hereby is to enhance the existing architectures so that these architectures provide the basis for a more efficient, reliable, generic and robust design, development and evaluation of distributed dynamic cooperative systems that can be used in Cycle 3 of D3CoS for the D3CoS demonstrators.

2.7 About Interaction, Communication Architecture MTTs

In a dynamic, cooperative system, communication is vital and represents a technical as well as scientific challenge. In order to define the D3CoS communication architecture, a process similar to the one defined for the whole DCoS architecture is being followed.

In the first cycle, a survey of communication architectures for distributed dynamic cooperative systems has been made. Due to the amount of existing architecture and the necessity to understand their inner workings, the actual evaluation through network simulation of those architectures will be made during cycle 2 and will be followed by the definition of the D3CoS communication architecture.

In line with the network community habits, the architecture will be defined in reference to the OSI model with layers ranging from the physical layer to the transport layer. Most of the definition work will be made in the third layer (networking layer) of the architecture so as to be able to deploy it with several link layers depending on the specific needs of each domain (e.g.: range, energy consumption,...).

Finally, in the third cycle, the communication architecture will be implemented and deployed on the paparazzi UAV demonstrator for the final D3CoS demonstration.

3 Existing Architectures

Following are the collected state of the art DCoS architectures from the domains Automotive, Unmanned Air Vehicle and Manned Air Vehicle summarized which are available as MTTs within D3CoS. These architectures are used for the Simulation platforms to allow experiments and evaluations already in the first D3CoS cycle, as well as to define features and requirements towards an abstract DCoS architecture.

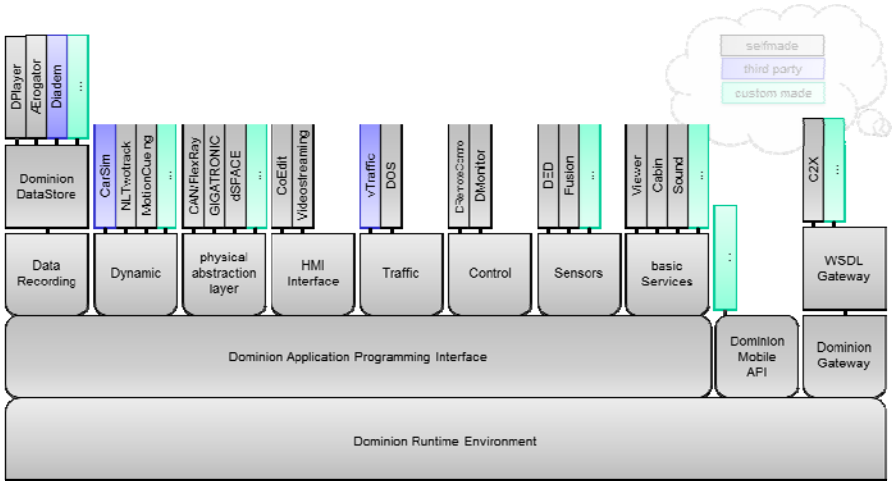
Each of the collected architectures is presented in at a subchapter and summarized by a table with the following elements:

Name:	Name of the architecture
Owner:	Developer and/or D3CoS user of the architecture
Domain:	Domains within this architecture is in use
Operation Area:	Short description of the field of use for this architecture
Perspective:	Possible architecture perspective e.g. on-board or traffic perspective
In use since:	Name / date of the first known implementation of this architecture
Current Implemented Version:	The currently available version (name, version, since date) of the implementation of the architecture
Current supported Operation Platforms:	Operation platforms of the current implemented architecture version
Project History:	Name of international and national projects (subtotal) where this architecture was in use before D3CoS
Used Concepts:	Architecture design concepts in use by this architecture (e.g. SOA)
Structure:	Graphical representation of the structure of this architecture
Data Exchange:	Methods how the data exchange is organized within this architecture
Communication of System Elements	Methods how are system elements (Tasks, Agents, Resources) connected at this architecture
Interfaces:	Interfaces /standards in use to connect the system elements of this architecture
Infrastructure / Hardware:	Target platforms / hardware of this architecture

3.1 DOMINION

DOMINION is a service-oriented architecture which follows the concept of service orientation, basic services like hardware access are encapsulated in reusable and loosely coupled services. The DOMINION implementation is a DCoS development and communication middleware which uses for any use-case (research facility) a certain set of base services for a flexible orchestration of new assistance and automation functions.

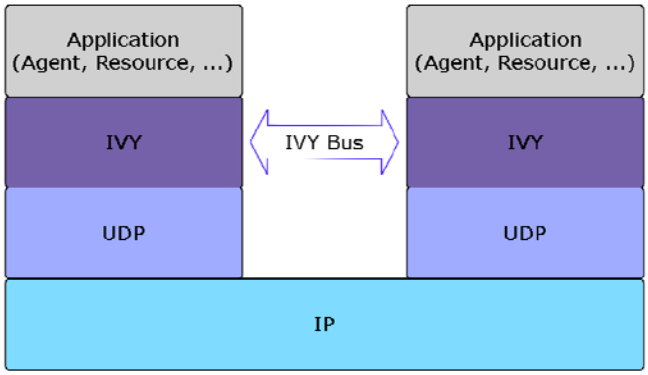
Name	DOMINION
Owner	German Aerospace Center (DLR) - Institute of Transportation Systems
Domain	Field of research: Automotive, Railroad System
Operation Area	Field of research: Design, development and analysis of Advance Driver Assistance Systems (ADAS) to run and evaluate them at simulators and prototype vehicles.
Perspective	On-Board perspective (Human - Machine ADAS)
In use since	DOMINION V1 since 2007
Current Implemented Version	DOMINION V2 since 2009
Current supported Operation Platforms	Windows XP SP3, Windows 7 32Bit, Linux Ubuntu LTS10.04
Project History:	International: • ISi-PADAS, HAVEit, InteractIVe National: • CarS, FAMOS, FIT, IMOST, RefelctAs, DeSCAS
Used Concepts	• Web Services [Schröder] • Asynchronous as well as synchronous runtime behaviour [Schindler] • Service Oriented Architecture (SOA) • Formal method definition • Formal data description For SOA, formal methods and formal data see: [Montenegro] [Köster] [Gacnik]

Structure	
Definition of Data Exchange	Via a virtual data bus (transparent to the user) This virtual bus is working on single PC and/or local network
Communication of System Elements	is loose organized with an asynchronous coupling
Interfaces	External • UDP → Standard: RFC 768 (1980) • TCP → Standards: RFC 793 (1981), RFC 1323 (1992) • CAN-Bus → Standard: ISO 11898 Internal • UDP → Standard: RFC 768 (1980) • XML → Standard: W3C XML-Specification • Ecore → Standard: EMOF
Infrastructure / Hardware	Standard PCs , Automotive PCs or Notebooks

3.2 IVY

IVY is a communication architecture based on the publish / subscribe concept. Data is published on a software bus and retrieved by application which subscribed for it in the form of regular expressions on the content of the messages.

Name	IVY
Owner	CENA (Centre d'Etudes de la Navigation Aérienne)
Domain	UAV / MAV
Operation Area	Used in interactive system development and in paparazzi UAV communications
Perspective	on-board an traffic perspective
In use since	IVYv1 1996
Current Implemented Version	IVYv3 2002
Current supported Operation Platforms	• Binary packages for Windows (win32, WinPocket PC), mac (OS X jaguar), Linux (packages for debian Etch, mandriva 10, red hat 6.2) and Sun OS (solaris 8) • Sources available in several languages (heavily tested under Linux debian)
Project History:	International: • Used in SESAR WP 4.7.2 experimentations • Used by paparazzi UAV system • Used by TEC team in Eurocontrol, for SkyTools project National: • Ivy is used by CENA within the Fugue Air Traffic Control simulator • Used by several laboratories and universities • Used by Intuilab (french company)
Used Concepts	• Publish-Subscribe communication • Broadcast and multicast IP capability

Structure	
Definition of Data Exchange	• Via IP packets sent to the IP address of the IVY bus • The IP address of the IVY bus can be multicast, no central control is needed
Communication of System Elements	• Applications need to *bind* to the IVY bus • Senders *publish* data to the bus via the API • Receivers *subscribe* to data form the bus and are notified asynchronously when it is available
Interfaces	• UDP Standard: RFC 768 (1980) • IVY offers a simple API available for the most common languages (C, C++, Java, python,...)
Infrastructure / Hardware	• Standard PC / workstations • If used distributed over several hosts, needs an multicast capable IP network

3.3 A-Globe

A-Globe architecture provides basic services and environment for running agents. The architecture provides the same level of services as JADE, COUGAAR, FIPA-OS and JACK. A-globe is not fully FIPA compliant.

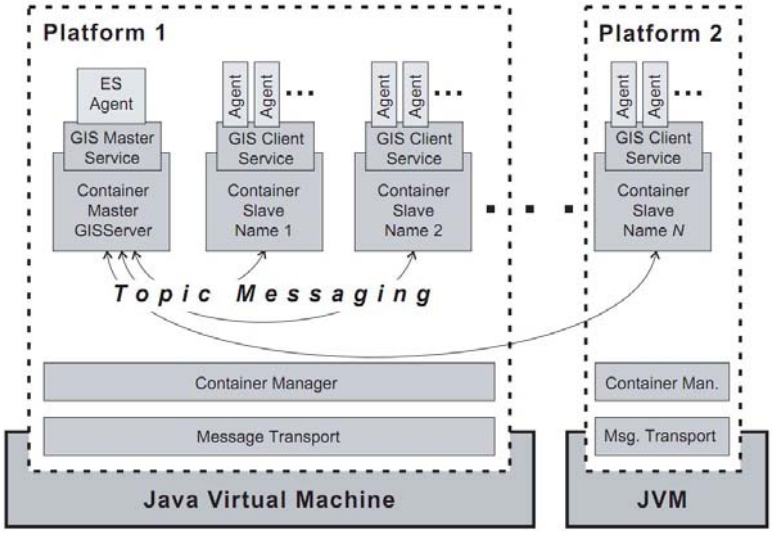
A-Globe is designed for testing experimental scenarios featuring agent's position and communication inaccessibility, but it can be also used without these extended functions. A-globe provides functions for the residing agents, such as communication infrastructure, store, directory services, migration function, deploy service, etc.

The system integrates one or more agent "platforms". Main parts of the architecture are:

- Agent platform – provides basic components for running one or more agent containers (i.e. container manager and message transport layer)
- Agent container – skeleton entity of A-globe, ensures basic functions, communication infrastructure and storage for agents
- Services – provide some common functions for all agents in one container
- Environment simulator (ES) agents – simulates the real-world environment and controls communication accessibility among other agent containers
- Agents – represent basic functional entities in a specific simulation scenario.

Name:	A-GLOBE
Owner:	CTU
Domain:	Agent-based research, UAV, MAV simulation, traffic simulation
Operation Area:	Short description of the field of use for this architecture
Perspective:	Can be used as a communication middleware or simulation tool for both perspectives
In use since:	2004
Current Implemented Version:	Version 5.5
Current supported Operation Platforms:	All Java virtual machine compatible platforms (Windows, Linux, MAC, embedded systems with Java)

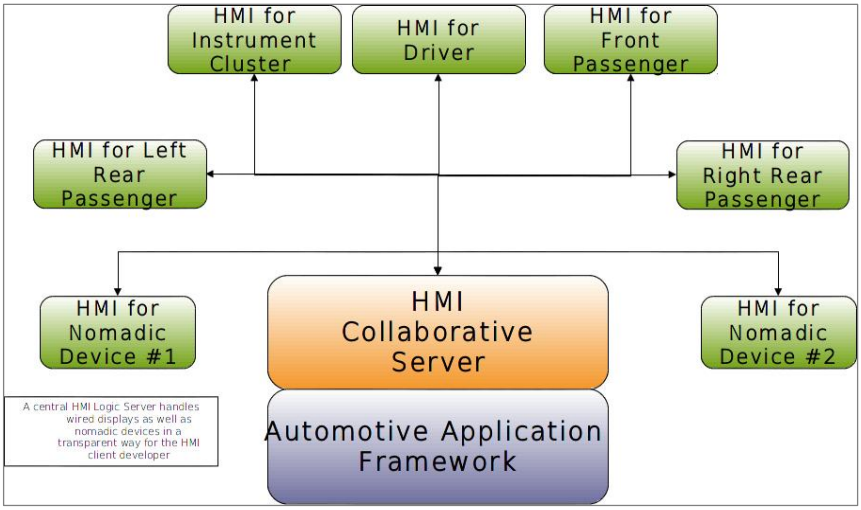
	D3CoS Designing Dynamic Distributed Cooperative Human-Machine Systems	

Project History:	Camnep(s), AgentFly(s), TacticalAgentfly used by industry: DENSO, Cadence, FL3xx
Used Concepts:	Service Oriented Architecture (SOA)
Structure:	
Data Exchange:	Data should be exchanged in a content of a message/topic. Data are serialized or externalized to xml for the transmission.
Communication of System Elements	Two kind of messages – 1) agent to agent messages, 2) system messages (i.e. topics). Topics are used by system services.
Interfaces:	Java API, java accessible communication interfaces - mainly TCP/IP, UDP
Infrastructure / Hardware:	PCs, embedded systems with java virtual machine

3.4 UI Cockpit

The architecture behind the UI-Cockpit allows to design and implement distributed embedded HMI applications easily. A concept used by this architecture is the MIB (Management Information Base) which allows configuring and extending components by modelling them via a XML interface. Additionally the architecture provides an abstraction layer which allows connecting specific low level components easily to reduce implementation time of new technologies and modules. UI-Cockpit therefore is a portable software development kit (SDK) which provides Application Programming Interfaces (APIs) for HMI components (Generic APIs to load and connect HMI modules, Specific APIs which depend on the considered graphical environment such as Altia, TAT, Qt…) and allows the developers to continue development with their favourite technology (e.g. Flash, QML, HTML5) by providing many existing connectors. UI-Cockpit uses the Model-View-Controller and Model-View-Presenter are both Design Patterns for HMI which is currently state of the art.

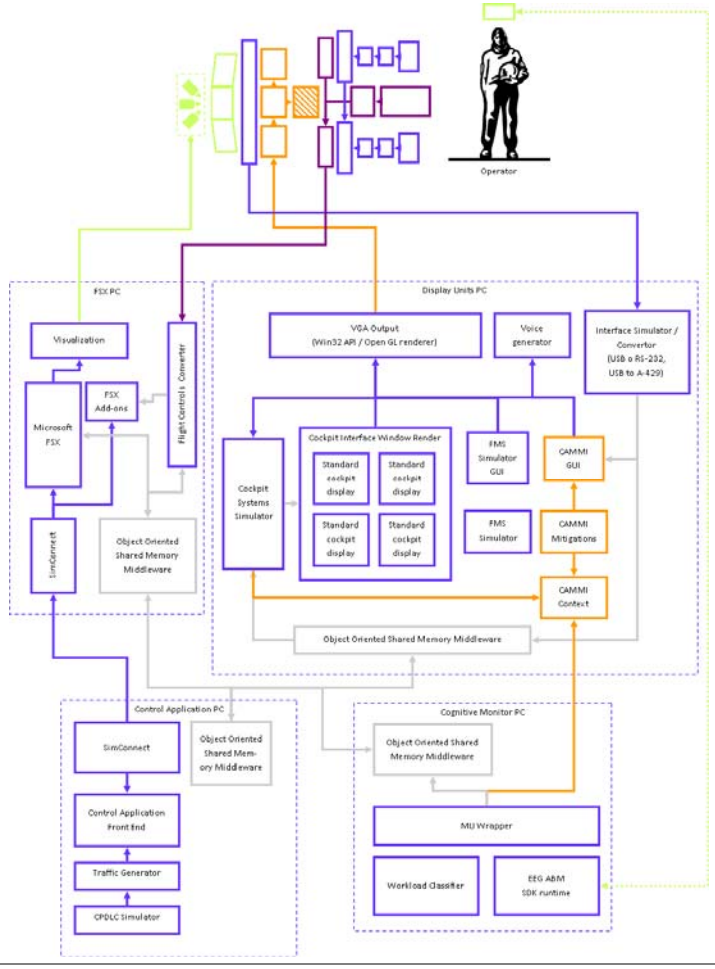
Name	UI Cockpit - Architecture for collaborative and distributed HMI
Owner	Visteon Software Technologies
Domain	Automotive
Operation Area	To develop an ecosystem to enable a quick implementation of distributed and collaborative HMIs within the car (on the vehicle displays and on CE devices) and to naturally extend the ecosystem for car-to-car environments
Perspective	On-Board perspective (Human - Machine ADAS - Infotainment)
In use since	UI.Cockpit v1.0 since 2011
Current Implemented Version	UI.Cockpit v1.1 since 2012
Current supported Operation Platforms	Linux Windows 7 QNX DI KERNEL.
Project History:	Already used internally for 2 years. Especially used by several car manufacturers on advance projects

Used Concepts	MIB (Management Information Base) MVC (Model/View/Controller), MVP (Model/View/Presenter), SNMP (Simple Network Management Protocol)
Structure	
Definition of Data Exchange	Data is exchanged between the server and the clients. Communication is bi-directional. Connection can be hardwired or wireless and various protocols can be used (new ones can be defined and loaded dynamically) Working on single hardware platform and/or local network
Communication of System Elements	Asynchronous Protocol can be dynamically loaded
Interfaces	External • C/C++ communication API for external clients: → Standards: ISO/IEC 14882:2003 • TCP/IP Sockets → Standards: RFC 793 (1981), RFC 1323 (1992) Internal • TCP/IP Sockets → Standards: RFC 793 (1981), RFC 1323 (1992) • C/C++ → Standards: ISO/IEC 14882:2003 • XML → Standard: W3C XML-Specification
Infrastructure / Hardware	x86, ARM, PowerPC

3.5 TRACS

TRACS is a simulation infrastructure and tool set aimed for integration of avionics software prototypes into cockpit simulation environment. The TRACS architecture is based on inter-process communication utilizing shared memory and sharing via Ethernet. Information is shared amongst the various PCs via the use of shared memory objects. Although this is the core method for sharing and distributing data within TRACS, there are also specialized mechanisms provide for data transmission and display. These mechanisms are provided to support real-time and pseudo-real-time embedded applications and displays. It is simulating avionic buses communication style and offers variety of tools for configuration and control of distributed simulation and data logging and analysis. It enable is with rapid advanced cockpit simulations integrating real avionic software with prototypes with different level of fidelity.

Name	TRACS Cockpit Simulator - Tool for Rapid Advanced Cockpit Simulation
Owner	Honeywell International s.r.o.
Domain	Aerospace (Advanced Technology R&D application)
Operation Area	• Enable early evaluation of Human Factors aspects of new prototypes and procedures • Evaluation of Technology Readiness Level before real HW prototype is done (precertification evaluation) • Pre-integration and testing of real avionic code • Prototyping Primary Flight Displays (PFD) and Multifunctional Flight Displays (MFD) displays for rapid HMI evaluations and mockup development
Perspective	On-board man-avionic systems and equipment
In use since	Since January 1999
Current Implemented Version	V4
Current supported Operation Platforms	Windows XP, 7 (32b and 64b)
Project History:	Integration

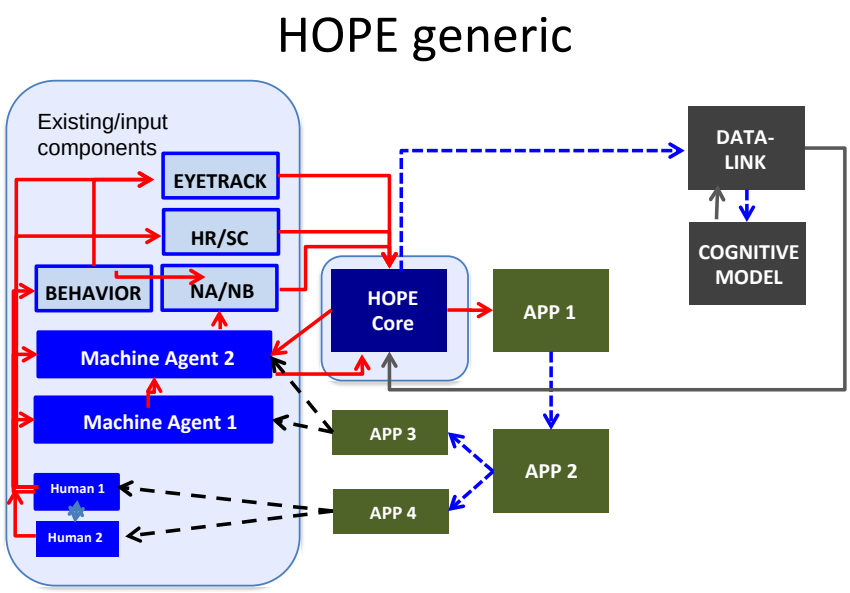
	• EPIC Business Jet glass cockpit system • NASA R&D projects • SESAR Joint Undertaking National • TACR – ATIS project
Used Concepts	• Real-time Distributed simulations • Avionic buses simulation (ASCB) • Shared Memory and Inter-process Communication • Open GL ES display prototypes • Object definition language • Internet object oriented communication engines • Computer software middleware platform
Structure	

Definition of Data Exchange	Communication using shared memory object described by meta data description files (shared memory object structure implemented by each component) Data flow configured by XML communication scheme
Communication of System Elements	Distributed real-time simulation and visualization (time is not synchronized between software components) Loose, asynchronous coupling using proprietary protocol for communication between components
Interfaces	External • TCP → Standards: RFC 793 (1981), RFC 1323 (1992) • UDP → Standard: RFC 768 (1980) • HLA → IEEE Standard 1516 • DIS → IEEE Standard 1278 • Object oriented internet communication engine Internal • Distributed shared memory objects
Infrastructure / Hardware	Working on single PC and/or local network / High-performance PC, Honeywell display units, MKB and CCD is used for HMI

3.6 HOPE

HOPE is a generic and conceptual architecture for augmenting inter-agent SA by catering simple algorithms to detect agent performance deviation from baselines and to consequently facilitate the identification of overall DCoS shortfalls and to propose task redistribution requirements and shared authority. HOPE is modular in order to incorporate other MTTs (e.g. PED/CASCAS) and can be extended or restricted as needed and achievable.

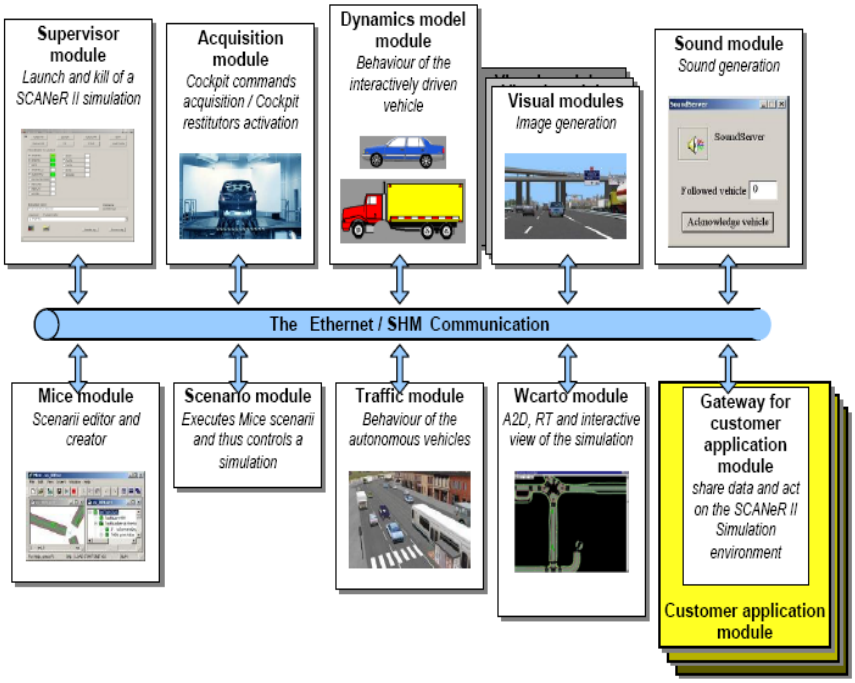
Name	HOPE - Human On-board Performance Embedding
Owner	truestream aerospace human factors
Domain	Field of research: Manned Aeronautics
Operation Area	Field of research: • design, development and analysis of future flight deck system applications • design, development and analysis of HIL-experiments • enabler for learning processes within a DCoS
Perspective	On-board perspective (Human-Machine/Machine-Machine)
In use since	V00: Original version since D3CoS proposal phase 2010
Current Implemented Version	V01: Updated, use-case specific version since start of D3CoS cycle 1 experiments in 2011
Current supported Operation Platforms	TBD (according to oncoming requirements)
Project History:	N/A (invented for D3CoS)
Used Concepts	• common auto-flight system architectures (e.g. Airbus & Boeing) [Boorman] • formal cognitive task models (e.g. PED) • behaviour deviation detection algorithms (from SOP/BASELINE) • MIDAS [Gore] • workload peak detection (e.g. PED or MIDAS)

Structure	
Definition of Data Exchange	According to requirements, capabilities and scope of DCOS (e.g. differing by manufacturer and platform – simulator vs. real aircraft with ACMS real-time telemetry.
Communication of System Elements	Dynamic task, resource and authority allocation functions are supported
Interfaces	External: • ARINC 429 (ARINC Standard 429) [ARNIC] • ARINC 629 (ARINC Standard 629) • AFDX (IEEE 802.3/ARINC Standard 664) • CAN bus (ISO 11898) Internal: • Shared memory of ACMS, FMGS and ACARS interface
Infrastructure / Hardware	Standard PCs, Notebooks, Electronic Flight Bags (EFBs), Flight Deck MCDUs/ACARS MUs, Flight Deck DUs

3.7 Static Driving Simulator

CoopSim is a static driving simulation platform, suitable for simulating a multi-agent complex environment. The DCoS concept system detects the situation in advance and performs a twofold action: inform drivers suggesting the right manoeuvres and interacts directly with the other agents, for an automatic resolution of the conflicts. In the meanwhile, the vehicles in the area (within a certain space related to the influenced of V2I and V2V running communication exchanges) will cooperate in order to not create any other unexpected unsafe influence to the safer manoeuvre suggested to the vehicles in critical position. Examples of Use Cases are the lane-changing or the Merging manoeuvre.

Name	CoopSim – Cooperative static driving simulator
Owner	Landi Renzo S.p.A. (LND)
Domain	Automotive (for research purposes)
Operation Area	Supports the development of a static driving simulation platform to enable a quick implementation of distributed and collaborative multi agent cooperative systems
Perspective	Traffic Perspective
In use since	V1.0 since 2006
Current Implemented Version	V2.22 since 2009
Current supported Operation Platforms	Windows 2000 / XP / Vista
Project History:	• ISI-PADAS • AIDE
Used Concepts	• Driver behaviour analysis in visual and cognitive distraction conditions [Tango] [Benedetto] • Automation effects on vehicle control [Tango2]

Structure	Static driving Simulator is based on a SCANeR II driving simulation engine. The following figure illustrates SCANeR II's distributed architecture concept::  Most of the modules rely on a single software layer that provides common functions. Among those functions are, of course, the communication layer that enables each module to send and receive messages according to the SCANeR II protocol.
Definition of Data Exchange	via a Shared Memory, working on single PC distributed over several PCs
Communication of System Elements	Asynchronous
Interfaces	External • Visual Module C++ API extension • AFS API extension • SCANeR API Internal • TCP/IP Sockets • C/C++
Infrastructure / Hardware	Standard PC, Notebooks

4 Architecture Requirements

In this chapter we describe which requirements should be employed for building the DCoS Architecture. Using our knowledge of the project we generated basic set of requirements / features we think that the Architecture has to satisfy / have.

It has been also collected broad set of requirements within work package no. 1.2. The Architecture is an MTT therefore all requirements from work package 1.2 should be applied. This set of requirements has been analyzed. Reasons why a requirement we considered to be non-relevant for the Architecture are described. For this purposes we used topical version of requirements from SharePoint.

We have also analyzed requirements for demonstrators described in the deliverable D1-01. It is important to guarantee that the Architecture will be efficiently usable in the demonstrators as typical instances of DCoS systems.

Finally, we collected requirements which had been used for creating state-of-the-art platforms.

All the requirements described above we considered as important and they will lead the development of the Architecture.

4.1 Requirements for abstract Architecture - Vision

Using knowledge of the project and vision of the target Architecture we defined very general set of requirements that shall be satisfied by the Architecture:

The abstract Architecture for DCoS shall

- be reusable
- be described by standard methods (UML-like)
- provide communication infrastructure for the agents

- support simulation, hardware-in-the-loop simulation (emulation), and hardware implementation
- support real-time properties
- be able to interact with design patterns from SP3 and other MTTs
- improve industrial processes (as an MTT)
- be independent on programming languages

4.2 Requirements from the Project

The SP1 produced requirements for demonstrators [D1-01] and for MTTs [D1-02]. It is clear that both these types of requirements can be relevant for DCoS Architecture.

Requirements for demonstrators are relevant because the Architecture has to be used to develop demonstrators in order to validate added value of the Architecture. Therefore, the Architecture must be also compatible with the requirements for demonstrators.

Requirements from WP1.2 are also relevant since the abstract Architecture is one of the MTTs and these requirements have stronger relationship with the Architecture because these requirements are domain-independent and more general for future DCoS systems

Result of the analysis will drive us to choose components and properties of the Architecture. Also, it will provide valuable feedback to SP1 and the requirements can be refined according to the analysis.

4.2.1 WP1.1 Demonstrator Requirements

The complete analysis tables for the demonstrator requirements are available in the Appendix 8.1 of this document. There we analyzed all demonstrator requirements from the different domains towards their use for the definition of DCoS architectures.

Following are some examples from the full demonstrator requirements tables available at the appendix:

SP1_MAV_GEN_REQ06_v01	This requirement is possible to adapt: the Architecture shall be able to handle real-time critical situations

SP1_GEN_REQ20_HW_CRF_v01	This requirement is possible to adapt: The Architecture shall support different interfaces and different kinds of networks.

SP1_AUT_REQ65_UC_DLR_v01	This requirement is AUT specific.

The left column shows the ID of the requirement which allows finding the requirement within the deliverable [D1-01]. The right column describes the result of the analysis of the demonstrator requirement within the WP3.4 architecture context.

4.2.2 WP1.2 MTT Requirements

The requirements are almost compatible with the Architecture. In the table (see Appendix 8.2) we discuss only requirements which have to be analyzed with corresponding authors in order to proof correctness of the analysis.

Following are two analysis result examples from the full MTT requirement table available at the Appendix:

req_34_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.

req_63_spec_HON	This requirement is not applicable for the Architecture because it defines structure of textual requirements.

The left column shows the ID of the requirements which allows finding the requirement within the deliverable [D1-02]. The right column

describes the result of the analysis of the MTT requirements within the WP3.4 Architecture context.

4.3 Requirements from the Existing Platforms

The last resource of the requirements (and also architectures and design patterns) we identified are industrial platforms which are currently used in the industry and platforms used by research institutions. Such platforms are built using their specific architectures and specific requirements. These requirements will be analyzed and result will enrich set of requirements for the Architecture.

4.3.1 DOMINION

- DOMINON should be a platform for integrating different processes and services
- DOMINON supports a formal method of modeling the data
- DOMINON supports a formal method of defining processes and services
- DOMINON is platform independent (program language/ operation system)
- DOMINON has a process runtime management
- DOMINON runs on a single machine
- DOMINON runs also as distributed system
- DOMINON provides a possibility to monitor the current available set of data of a DOMINION setup
- DOMINON provides a configurable data recording
- DOMINON allows to replay recorded data in a custom way
- DOMINON provides a method for configuring an controlling a DOMINION setup
- DOMINON is able to run in realtime
- DOMINON supports the development of new applications with provided tools
- DOMINON reduces the development complexity
- DOMINON should only need a loose coupling between processes and services

- DOMINON allows an easy integration of third party hardware and software modules
- DOMINON allows to integrate standard interfaces
- DOMINON enables the design, development and evaluation of advanced driver assistance systems
- DOMINON has clocked processes / services
- DOMINON uses asynchrony data exchange
- DOMINON enables a continuous development process

4.3.2 Ivy

- IVY shall allow for easy prototyping of interactive systems
- IVY shall allow for publish/subscribe communication
- IVY shall allow for direct messages sent between applications
- IVY shall be independent from platform and computer language
- IVY shall work on a single computer without network interface as well as distributed over a network
- IVY shall use standard communication services
- IVY shall minimize network resource usage
- IVY shall have an open-source, non-contaminant license

4.3.3 A-Globe

- A-Globe shall be multi-platform
- A-Globe shall support agent lifecycle including agent migration and agent's state migration over the network
- A-Globe shall provide communication infrastructure
- A-Globe shall provide communication monitoring tool
- A-Globe shall support agent conversations and protocols
- A-Globe shall provide asynchronous running of agents
- A-Globe shall be deployable in a distributed manner
- A-Globe shall provide communication inaccessibility support

4.3.4 UI Cockpit

- UI Cockpit shall implement separation of functional layers and GUI/HMI layers
- UI Cockpit support generic software and interfaces for different embedded system operating systems (e.g. Linux/Moblin, MS Auto, QNX, Android)
- UI Cockpit should run at least on Windows and Linux
- Resources files, multimedia files and config files can be local, remote, or accessible through configurable and extensible connectors
- UI Cockpit architecture shall be distributed amongst various machines, their number can be extended/reduced
- All configuration files use human readable xml format
- the developer should only concentrate on the development of new applications and not the connection to other modules (e.g. encapsulate the whole data interchanging mechanism)
- it is necessary to be able to integrate new hard- and software to a UI Cockpit setup
- for integrating new modules there has to be a way to integrate standard protocols and interfaces
- no process should be slowed down due to new input data that is not provided by another process in time

4.3.5 TRACS

- TRACS simulation infrastructure should allow testing of display products (more generally avionic products)
- TRACS should enable rapid prototyping and fast integration of display products without extensive change of products code due to different communication means required by simulation bus
- TRACS should allow easy integration of new prototypes of displays and avionic technology
- TRACS should stay as close as possible to real avionic buses from display's point of view
- TRACS should allow emulation of ASCB communication buffer formats and timing (bus for avionic in BGA)

- TRACS should allow simulation of real time communication (time triggered communication, frequency of information exchange)
- TRACS should enable distributed simulation environment (simulation of avionic between multiple computers connected via network)
- TRACS should operate on MS Windows operation system
- TRACS should allow configuration management and versioning during development
- TRACS should allow easy extensibility of values on simulation buses
- TRACS should support debugging, on-line capture of data, data logging etc.
- TRACS should be robust to fault of one simulation component and wrong configuration

4.3.6 HOPE

- HOPE shall allow to let agents interact
- HOPE shall accommodate identifiable agents/components
- HOPE shall allow for the integration of different databases (different types of a/c's, different types of procedures - modular enhancement of the baseline) to enhance PED to form a universal task model
- HOPE shall provide the architecture that makes different flows of tasks between agents, different inputs from resources and different applications of procedures comparable, e.g. across different a/c's or data once flown in the aircraft – compared to data collected in SIM
- HOPE shall allow for an easy analysis/assessment structure (i.e. of data collected during the experiments)
- HOPE shall allow for an easy analysis of task distribution, e.g. in case of a reduction of crew (Single Pilot Operation – SPO) – how would this affect the workload of the remaining pilot in case there wouldn't be any additional assistance systems (that could take over most of the additional tasks) on board
- HOPE's architecture shall allow a modelling tool to easily re-allocate tasks to different agents and to let models simulate the changes in workload and visualize the respective effects
- HOPE shall cater at least 2 human agents
- HOPE shall cater at least 2 machine agents

- HOPE needs the implementation of learning algorithms to baseline individual agent behaviour
- HOPE shall be able to accommodate TARF configurations of different domains

4.3.7 Static Driving Simulator

- CoopSim should be able to model the cooperation, coordination, conflict, allocation, distribution, authority, cooperation modes among agents
- CoopSim should be configurable to take into consideration new requirements (or modification of the existing requirements) during the Project Cycles
- CoopSim should allow the developer to seamlessly develop and integrate the DCoS in a single platform
- CoopSim should provide the developer with an environment to simulate the behavior of the agent before testing it in the real environment
- CoopSim should provide the developer with a graphic environment to select the characteristic of the system he/she is going to develop and then create a software framework according to these preferences
- CoopSim should allow the development of real-time solutions
- CoopSim shall be able to work in a large variety of scenarios, such as: highways, urban and extra-urban roads, etc...
- CoopSim shall be operative for ego-vehicle speed in the range (0-150) km/h, that is (0-41.7)m/s
- CoopSim shall handle time critical situation

5 Abstract DCoS Architecture Vision

This chapter will present the first steps toward the definition of the abstract DCoS architecture. We will start by presenting the generic DCoS framework and how it instantiate in the on-board and traffic perspectives. We will then present the features of one of the major modules of the abstract architecture, the communication architecture.

5.1 Generic DCoS Framework

In [Lüdtke], a generic framework for DCoS is presented. This framework, shown in Figure 4, defines a DCoS (1) as being composed of a set of agents, human (2a) or machine (2b).

The communications and interactions (3) between the agents creates a cooperative network. Tasks (4) are assigned (6) to the agents who fulfil them by accessing (7) to resources (5). The cooperative system as a whole acts on one or more controlled objects (8) while being immersed in an environment (9).

It is this generic framework (Figure 4) that will be used to define the final abstract DCoS architecture which goal is defined in WP3.4 as:

> *An abstract DCoS architecture defines interfaces for system elements (agents and resources) and describes their behaviour in dynamic distributed cooperative systems for the four domains addressed in the D3CoS project (MAV, UAV, MARITIME, AUT).*

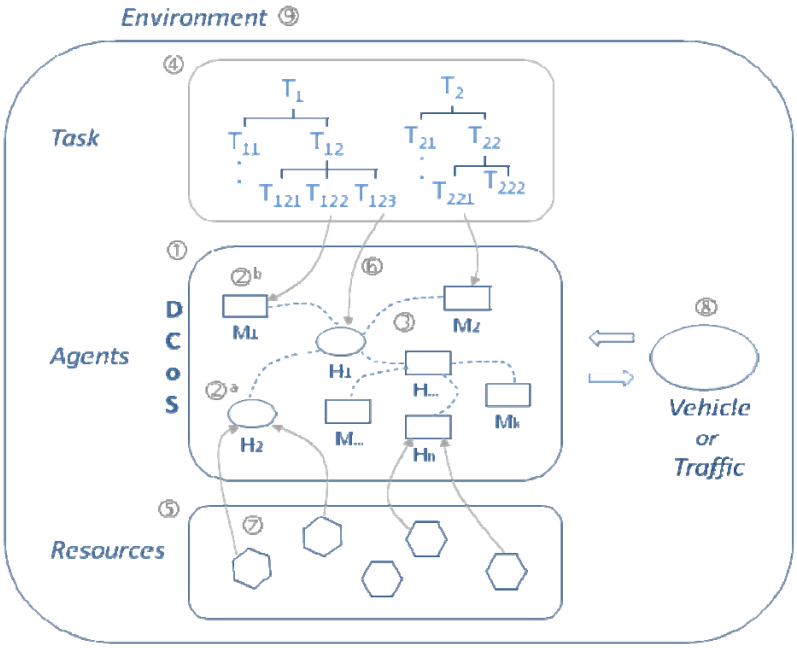


Figure 4: Generic DCoS Framework

In the next section we will see how the described framework instantiates itself in the envisioned on-board and traffic perspectives.

5.2 Traffic and On-Board Perspective

Use cases, scenarios and demonstrators have been deliberately divided into the traffic and the on-board perspectives. Obviously, it corresponds to different needs. First is related to the optimisation of the traffic flow involving several vehicles and the second one focuses on the driver/pilot surroundings (car/cockpit). But as reflected by the choice of the terms, these are just different *perspectives* and the same abstract architecture could be used to cover both points of view.

As suggested by the Figure 5 below, the traffic perspective can be perceived as a higher level view, whereas the on-board perspective zooms into the driver closer ecosystem.

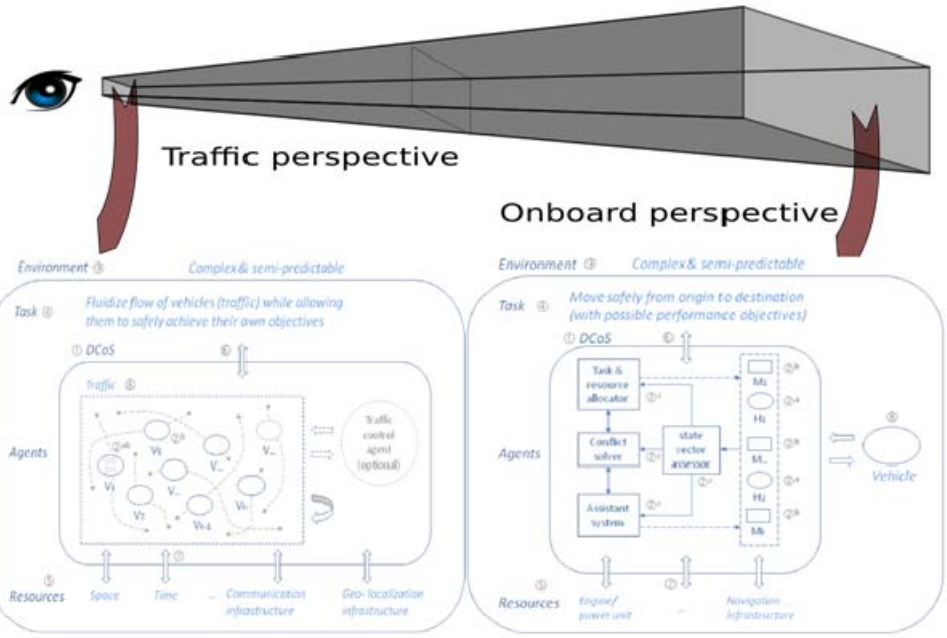


Figure 5: Traffic and On-Board perspectives

What is important here is that the *nature* of the interactions for each perspective can be described and modelled by the same semantic and the same abstract architecture.

For instance a vehicle with a human - machine system can be described by on-board perspective architecture. This architecture deals with agents like a lane change ADAS connect them with resources such as the steering wheel or displays at the instrument cluster and allows to perform tasks like calculating, how to reach a good position to change the lane.

If the same vehicle has also the possibility to communicate via a communication infrastructure as resource with other vehicles nearby (the vehicles are the agents) to find together a good position for a lane change (task) or even to create a good lane change position cooperatively by a slightly braking of a vehicle on the target lane than this distributed system will be described by a traffic perspective architecture.

Due to that both perspectives uses the same semantic elements: agents, resources and tasks can an abstract architecture describe both systems if

her definition of the semantic elements is comprehensive and flexible enough to reflect various levels of states and interactions.

The Figure 6 and Figure 7 show the instantiation of the generic DCoS Framework presented in the introduction for the on-board and traffic perspective respectively.

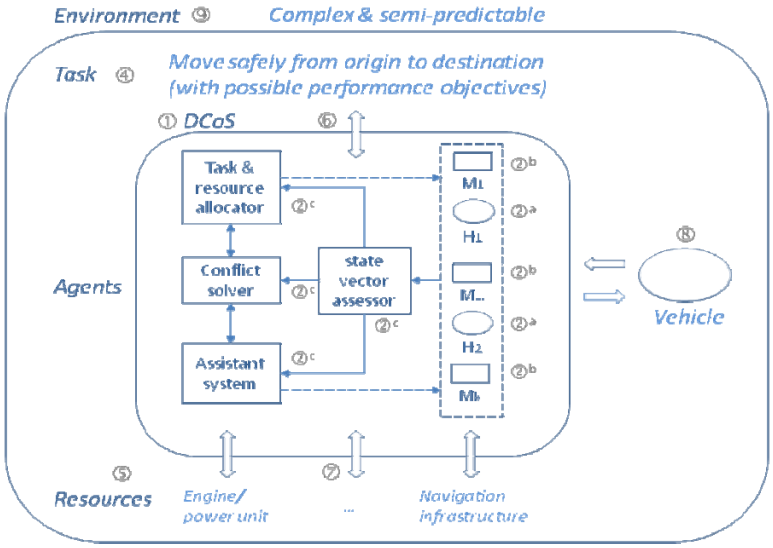


Figure 6: DCoS On-Board Perspective

On-board perspective: Figure 6 a vehicle (e.g., a car, an airplane, a ship) (8) is controlled by a cooperative system made of on-board human (2a) and machine (2b) agents. They are assisted by specialized "assisting" machine agents (2c) (state assessor, conflict solver, a task & resource allocator, assistant system).

Traffic perspective: Figure 7 multiple vehicle agents, possibly complemented by an (optional) control agent, external to the vehicles (e.g., air traffic control).The vehicles are either machine agents (2b) (e.g., an unmanned aircraft) or simpler DCoS (2ab), made of human and machine agents (e.g., a car) (link with the on-board perspective). The object under control (4) is the flow of the vehicles itself, or traffic.

In both perspectives, the vehicles and the agents are immersed in a typically complex and semi-predictable environment (9) (e.g., weather, other vehicles, uncontrolled traffic).

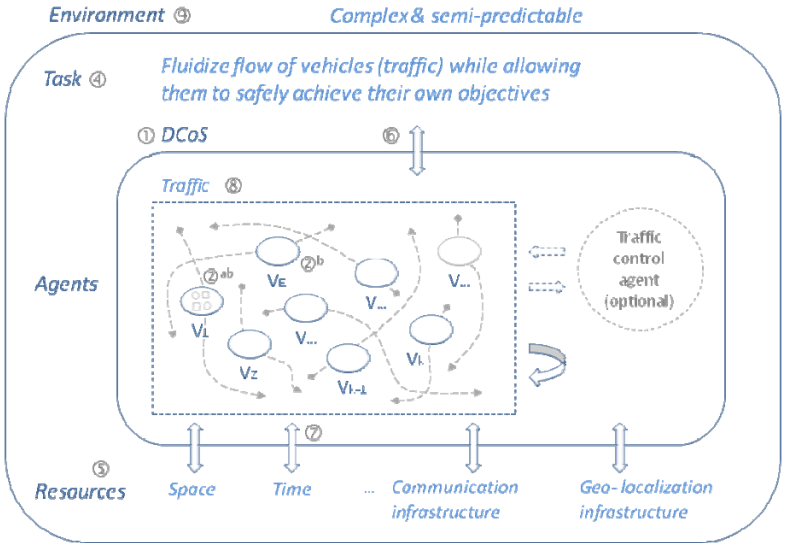


Figure 7: DCoS Traffic Perspective

5.3 Communication Architecture Features

To perform a DCoS mission, a specific communication architecture should be deployed. This architecture has to be able to adapt to the needs of dynamic, cooperative, distributed agents. It is responsible for managing communications between agents, that is primarily to forward messages between agents. In addition, the cooperative nature of the agents may require that messages be transmitted with certain characteristics in terms of reliability, delay and throughput. Those characteristics form the quality of service (QoS) of the communication.

In a DCoS, agents may be mobile and move according to their tasks' needs in different directions causing modifications of the network topology. The mobility and the distribution of the communicating agents thus impose the adoption of a communication architecture with the characteristics of a mobile ad hoc network.

Mobile Ad hoc networks (MANET), are self configuring, self healing networks designed to allow a set of mobile nodes to communicate with each others over wireless links in the absence of any infrastructure or

centralised administration. As mobile nodes have a limited transmission range, nodes that lies within transmission range of each other can communicate directly, while other intermediate nodes are needed to forward messages between nodes that are further away.

In order to discover the network topology, and to find the best route for each destination (i.e.: the successive nodes from the source of data to its destination), a routing protocol is used. Its role is to define the best route from the source to the destination eventually taking into account the quality of services aspects of the communication.

The choice of the suitable routing protocol for the dynamicity of the DCoS network architecture is very important to guarantee the satisfaction of agents' needs. The chosen protocol should be able to forward information to the destination following three types of routing paradigms:

- unicast, which consists in sending data to the destination host based on its address;
- multicast, which consists in forwarding data from the sender to a group of destination hosts that share an address;
- anycast which consists in forwarding the data to the closest host configured with a given address.

Apart from the needs to send data to destinations, the cooperative agents will require from the communication system to have their data forwarded with a Quality Of Service (i.e.: delay, throughput and reliability) that will be defined by application concepts such as the task being achieved, the type of information being sent (e.g.: video, audio, status information, etc...). The communication system is responsible for resources allocations in order to offer the required QoS.

Moreover, the communication system should track the connectivity of each agent and have this information accessible to all agents connected to the network. As an example, in the case of an imminent loss of connectivity, the network must inform the DCoS, which may use another agent as a relay in order to create a new route to the soon disconnected agent.

In a dynamic environment like DCoS, each agent has a specific task and role. Some tasks are more important than the others, or they must be accomplished first. Agents performing those tasks need smaller delay, and higher throughput to send their messages. Thus, different priority

may be defined between different agents depending on their missions. The network should be aware of different priority while managing communications: message sent from an agent with the highest priority should be forwarded to its destination first.

In order to accomplish their tasks, agents have to cooperate with each other by exchanging different type of messages: messages related to their tasks such as image, video, mission information, messages describing their personal conditions. Thus, it is also essential to define priorities among the different types of messages. Managing QoS is a big challenge for the DCoS communication system and will need to be dealt with methodologies similar to the DIFFServ [Blake] mechanisms. DIFFServ marks packets according to the service they should be given and then uses predefined queuing strategies to send those packets. This model could be used for the DCoS communication system to manage QoS between agents.

Those features are being taken into account in the definition of the communication architecture. The next step in this definition will be the release of the interface to the communication architecture. Then the architecture itself will be finalised and implemented for the demonstrations.

6 Conclusions and Next Steps

During the first D3CoS cycle (year) the goal of the work package 3.4 architectures, interfaces and communication infrastructure was to collect state of the art DCoS reference architectures and to support the collection of their most important requirements. Additionally are these collected DCoS architectures important to allow a closed D3CoS modelling loop already in the first year. Therefore the collected DCoS architectures were used to provide Simulation Platforms (WP3.2). These Platforms allowed performing of first D3CoS experiments already within the first year of the project.

Next to the collecting of the state of the art DCoS architectures the WP3.4 analysed the WP1.1 Demonstrator and WP1.2 MTT requirements. The thereby as useful identified requirements to describe DCoS architectures are the most important requirement to define an abstract DCoS software architecture within the second D3CoS cycle. Additionally provides the WP3.4 feedback to the SP1 about the analysed requirements to support the D3CoS requirement table improvement process.

In parallel to the collecting and analysing tasks, the WP3.4 worked on the definition of vision about: how to define an abstract DCoS software architecture, what is the goal of an abstract DCoS software architecture, how to handle the traffic and on-board perspectives or how to define the communication architecture for an abstract DCoS architecture.

Some of these points, like how to deal with the perspectives or a first definition of the communication architecture by their features are already described by this deliverable (see chapter 5). Overall is the definition of the architecture vision not completed jet and will be continued at the beginning of cycle 2.

Therefore the next WP3.4 steps are: Finalizing of the requirement analysis as well as the completing of the definition of the abstract architecture vision. Afterwards to continue by the definition and description (UML) of abstract DCoS software architecture and their relationships like described in chapter 2. This requires to seek the

commonalities between the different architectures to integrate them in the common abstract D3CoS architecture.

7 References

D3CoS Deliverables:

[D1-01] D3CoS D1-01 (2011); Requirements for Demonstrators – Initial Version

[D1-02] D3CoS D1-02 (2011); Requirements for MTTs – Initial Version

[D2-01] D3CoS D2-01 (2012); Demonstrators – Preliminary Version

[D3-01] D3CoS D3-01 (2011); Generic DCoS Framework - Preliminary Version

[D3-02] D3CoS D3-02 (2012); Modelling and Simulation Platform

[D3-03] D3CoS D3-03 (2012); Reference Designs and Design Patterns for Cooperation & DCoS State Inference and Adaptation - Preliminary Version

[D3-04] D3CoS D3-04 (2012); Reference Design and Design Patterns for Multi-Modal Human Machine Interfaces - Preliminary Version

[D3-06] D3CoS D3-06 (2012); DCoS Methodology – Preliminary Version

References and Papers:

[Balzert] Helmut Balzert: *Lehrbuch der Software-Technik*. 2. Auflage. Spektrum Akademischer Verlag; Heidelberg: 2001; ISBN 3-8274-0301-4.

[Benedetto] Simone Benedetto, Marco Pedrotti, Luca Minin et al. *Driver workload and eye blink duration*, 199-208. In Transportation Research Part F: Traffic Psychology and Behavior 14 (3), 2010.

[Blake] S. Blake et al., RFC 2475 : *An Architecture for Differentiated Services*, December 1998

[Boorman] Boorman, D.J., Mumaw, R.J: *A New Autoflight FMS Interface: Guiding Design Principles*; Seattle: 2007, WA: Boeing Commercial Airplanes

[Gacnik] J.Gacnik, O. Häger, M. Hannibal: *A service-oriented architecture for the human centered design of intelligent transportation systems*, in European Conference on Human Centered Design for Intelligent Transport Systems, Lyon: April 2008

[Gore] Gore, B. F.: *Man-Machine Integration Design and Analysis System (MIDAS) v5: Augmentations, Motivations, and Directions for Aero-nautics Applications,* in P. C. Cacciabue, M. Hjalmdahl, A. Luedtke and C. Riccioli (Eds.). Human Modelling in Assisted Transportation; Berlin: Springer, 2010

[Köster] F. Köster, J. Gacnik, M. Hannibal: *Serviceorientierung als Zugang zur Strukturierung von In-Vehicle Softwaresystemen*, im 4. Braunschweiger Symposium: IMA 2008 Informationssysteme für mobile Anwendungen, Braunschweig: 2009

[Lüdtke] A. Lüdtke et al., *Designing dynamic distributed cooperative Human-Machine Systems*, in Work: A Journal of Prevention, Assessment and Rehabilitation, vol 41, p 4250-4257, Jan. 2012

[Montenegro] S. Montenegro, F. Dannemann, L. Dittrich, B. Vogel, U. Noyer, J. Gacnik, M. Hannibal, A. Richter, F. Köster: *(SpacecraftBusController+AutomotiveECU)/2=UltimateController*, in Lecture Notes in Informatics (GI-Edition), 160, Page 103-114. Gesellschaft für Informatik. Software Engineering 2010 / ENVISION2020, Paderborn: February 2010

[Schindler] J. Schindler, C. Harms, U. Noyer, A. Richter, F. Flemisch, F. Köster, T. Bellet, P. Mayenobe, D. Gruyer: JDVE: *A Joint Driver-Vehicle-Environment Simulation Platform for the Development and Accelerated Testing of Automotive Assistance and Automation Systems,* in Human Modelling in Assisted Transportation*;* Italia: Springer-Verlag, 2011 Srl. ISBN 978-88-470-1820-4,

[Schröder] M. Schröder, M. Hannibal, J. Gacnik, F. Köster, C. Harms, T. Knostmann: *Ein Labor zur modellbasierten Gestaltung interaktiver Assistenz und Automation im Automotive-Umfeld*, AAET 2010, Braunschweig, Februar 2010,. ISBN 978 3 937655 23 9

[Tango] Fabio Tango, Marco Botta, Luca Minin, Roberto Montanari: *Non-intrusive Detection of Driver Distraction using Machine Learning Algorithms*. ECAI 2010: 157-162

[Tango2] Fabio Tango, Luca Minin, Raghav Aras, Olivier Pietquin*: Automation Effects on Driver's Behaviour When Integrating a PADAS and a Distraction Classifier*. HCI (17) 2011: 503-512]

Web References:

[ARINC] ARINC Standards: www.arinc.com

[FIPA] FIPA, *FIPA Abstract Architecture Specification*; December 2012; http://www.fipa.org/

8 Appendix

8.1 WP1.1 Demonstrator Requirements

Following are the complete analysis tables for the demonstrator requirements available which are already introduced in chapter 4.2.1 A detailed description of the demonstrator requirements is available in the deliverable [D1-01]:

8.1.1 Manned Aeronautics (MAV) demonstrator requirements:

SP1_MAV_GEN_REQ01_v01	Not applicable. The stabilized approach is MAV specific.
SP1_MAV_GEN_REQ02_v01	It is not clear how to adapt this requirement for the Architecture
SP1_MAV_GEN_REQ03_v01	Not applicable. The stabilized approach is MAV specific.
SP1_MAV_GEN_REQ04_v01	Not applicable. The stabilized approach is MAV specific. There could be another requirement as “The Architecture shall support different scenarios”
SP1_MAV_GEN_REQ05_v01	It is not clear how to apply this requirement for the Architecture.
SP1_MAV_GEN_REQ06_v01	This requirement is possible to adapt: the Architecture shall be able to handle real-time critical situations
SP1_MAV_GEN_REQ07_v01	It is not clear whether is necessary to have a requirement: The Architecture should contain at least one human agent and one machine agent
SP1_MAV_GEN_REQ08_v01	It seems that this requirement is MAV specific
SP1_MAV_GEN_REQ09_v01	It seems that this requirement is MAV specific
SP1_MAV_GEN_REQ10_v01	It seems that this requirement is MAV specific
SP1_MAV_GEN_REQ12_v01	This requirement is possible to adapt: The Architecture shall allow for collection and storage of all data relevant for evaluation of the DCoS

SP1_MAV_GEN_REQ13_v01	This requirement is possible to adapt: The Architecture shall allow the user to override/ignore functions of the DCoS System
SP1_MAV_GEN_REQ14_v01	It is not clear how to adapt this requirement.
SP1_MAV_GEN_REQ15_v01	It is not clear how to adapt this requirement.
SP1_MAV_GEN_REQ16_v01	It is not clear how to adapt this requirement.
SP1_MAV_INTFC_REQ17_v01	This requirement is possible to adapt:The Architecture shall support interfacing data from other systems in real-time
SP1_MAV_INTFC_REQ18_v01	This requirement is possible to adapt: The Architecture shall support processing data from other systems in real-time
SP1_MAV_INTFC_REQ19_v01	This requirement is possible to adapt: The Architecture shall be support processing data from other systems in real-time
SP1_MAV_INTFC_REQ20_v01	This requirement is possible to adapt: The Architecture should support processing data from other systems
SP1_MAV_INTFC_REQ21_v01	This requirement is possible to adapt: The Architecture shall provide data updated and at a fixed rate
SP1_MAV_INTFC_REQ22_v01	This requirement is possible to adapt: The Architecture shall be able to support different interfaces for different kinds of networks
SP1_MAV_INTFC_REQ23_v01	This requirement is possible to adapt: The Architecture shall enable the agents to interact with each other via different interfaces
SP1_MAV_HMI_REQ24_v01	It is not clear how the Architecture can select the appropriate channel for the user
SP1_MAV_HMI_REQ25_v01	This requirement is possible to adapt: The Architecture shall support multimodal input/output
SP1_MAV_HMI_REQ26_v01	It is not clear how to adapt this requirement.
SP1_MAV_HMI_REQ27_v01	Not applicable, the requirement is MAV specific

SP1_MAV_HMI_REQ28_v01	This requirement is possible to adapt: The Architecture should provide information about a failure/limited availability within the system to the user
SP1_MAV_HMI_REQ29_v01	Not applicable, this requirement is HMI specific
SP1_MAV_HMI_REQ30_v01	Not applicable, this requirement is HMI specific
SP1_MAV_HMI_REQ31_v01	It is not clear how to adapt this requirement.
SP1_MAV_TECH_REQ32_v01	This requirement seems to be MAV specific.
SP1_MAV_TECH_REQ33_v01	It is not clear how to adapt this requirement.
SP1_MAV_TECH_REQ34_v01	It is not clear how to adapt this requirement.
SP1_MAV_TECH_REQ35_v01	Not applicable, this requirement is MAV specific.
SP1_MAV_TECH_REQ36_v01	It is not clear whether it is a cross-domain requirement
SP1_MAV_TECH_REQ37_v01	This requirement is possible to adapt: The Architecture shall support data format as much general as possible

8.1.2 Unmanned Aeronautics (UAV) demonstrator requirements:

SP1_UAV_REQ01_PERF_SGA_v01	This requirement is possible to adapt: The Architecture shall support large variety of scenarios
SP1_UAV_REQ02_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ03_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ04_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ05_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ06_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ07_PERF_SGA_v01	This requirement seems to be UAV specific.

SP1_UAV_REQ08_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ09_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ10_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ11_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ12_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ13_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ14_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ15_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ16_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ17_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ18_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ19_PERF_SGA_v01	This requirement seems to be UAV specific.
SP1_UAV_REQ20_PERF_SGA_v01	The same as SP1_UAV_REQ01_PERF_SGA_v01
SP1_UAV_REQ21_PERF_SGA_v01	It is not clear how to adapt this requirement.
SP1_UAV_REQ22_PERF_SGA_v01	This requirement is possible to adapt: The Architecture shall support emulation of several sensors
SP1_GEN_REQ23_PERF_CRF_SGA_01	This requirement is possible to adapt (similar to SP1_MAV_GEN_REQ06_v01): the Architecture shall be able to handle real-time critical situations
SP1_GEN_REQ24_HW_SGA_v01	This requirement is possible to adapt: The Architecture shall support different interfaces for different kinds of networks
SP1_UAV_REQ25_GEN_CTU_v01	Not applicable, this requirement is HMI specific
SP1_UAV_REQ26_GEN_CTU_v01	It is not clear how to adapt this requirement.

SP1_UAV_REQ27_GEN_CTU_v01	It is not clear how to adapt this requirement.
SP1_UAV_REQ28_GEN_CTU_v01	This requirement is possible to adapt: The Architecture shall be instantiated on HW
SP1_UAV_REQ29_GEN_CTU_v01	It is not clear how to adapt this requirement.
SP1_UAV_REQ30_GEN_CTU_v01	Not applicable, this requirement is UAV specific
SP1_UAV_REQ31_GEN_CTU_v01	It is not clear how to adapt this requirement.
SP1_UAV_REQ32_GEN_CTU_v01	It is not clear how to adapt this requirement.
SP1_UAV_REQ33_GEN_CTU_v01	Not applicable, this requirement is UAV specific
SP1_UAV_REQ34_GEN_CTU_v01	similar to SP1_MAV_GEN_REQ13_v01, adaptable
SP1_UAV_REQ35_GEN_CTU_v01	Not applicable, this requirement is UAV specific
SP1_UAV_REQ36_GEN_CTU_v01	It is not clear how to adapt this requirement. It seems to be UAV specific.
SP1_UAV_REQ37_GEN_CTU_v01	It is not clear how to adapt this requirement.
SP1_UAV_REQ38_GEN_CTU_v01	This requirement is possible to adapt: The Architecture shall support wide range of random perturbations
SP1_UAV_REQ39_PERF_RDE_VOI_v01	It seems to be UAV specific requirement.
SP1_UAV_REQ40_PERF_RDE_VOI_v01	It seems to be UAV specific requirement.
SP1_GEN_REQ41_SWARM_VOI_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ42_SWARM_VOI_v01	It is not clear whether this requirement is cross-domain. It seems to be reasonable to introduce requirement: The Architecture shall support adding agents.
SP1_UAV_REQ43_PERF_VOI_v01	It seems to be UAV specific requirement.
SP1_GEN_REQ44_SECURE_VOI_v01	This requirement is possible to adapt: The architecture shall support examination of agent activity retrospectively
SP1_GEN_REQ45_SWARM_VOI_v01	It is not clear how to adapt this requirement.

SP1_GEN_REQ46_INTERFACE_VOI_v01	This requirement is possible to adapt: The Architecture shall support multi-modal output (similar to SP1_MAV_HMI_REQ25_v01)
SP1_GEN_REQ47_INTERFACE_VOI_v01	This requirement is HMI specific.
SP1_GEN_REQ48_SECURE_VOI_v01	This is similar to: SP1_MAV_GEN_REQ13_v01, SP1_UAV_REQ34_GEN_CTU_v01, adaptable
SP1_GEN_REQ49_SECURE_VOI_v01	It is not clear how to adapt this requirement
SP1_UAV_REQ50_SWARM_RDE_VOI_v01	This requirement is UAV specific.
SP1_UAV_REQ51_SWARM_RDE_VOI_v01	This requirement is UAV specific.
SP1_UAV_REQ52_SWARM_RDE_VOI_v01	This requirement is UAV specific.
SP1_UAV_REQ53_PERF_RDE_VOI_v01	This requirement is UAV specific.
SP1_GEN_REQ54_SWARM_VOI_v01	It seems to be UAV specific requirement. It could be reasonable to introduce requirement “The Architecture shall support cooperative solutions of conflicts”.
SP1_UAV_REQ55_SWARM_VOI_v01	It seems to be UAV specific requirement. It could be reasonable to introduce requirement “The Architecture shall support agents with different sensors and capabilities”.
SP1_GEN_REQ56_SWARM_VOI_v01	It seems to be UAV specific requirement. It could be reasonable to introduce requirement “The Architecture shall support transferring of responsibility.”
SP1_UAV_REQ57_SWARM_RDE_VOI_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ58_PERF_RDE_VOI_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ59_SW_VOI_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ60_PERF_RDE_VOI_v01	It is not clear what the requirements means.
SP1_UAV_REQ61_PERF_RDE_VOI_v01	It is UAV specific requirement.
SP1_GEN_REQ62_SW_RDE_VOI_v01	It is UAV specific requirement.
SP1_GEN_REQ63_SW_RDE_VOI_v01	It is UAV specific requirement.
SP1_GEN_REQ64_SW_RDE_VOI_v01	This requirement is possible to adapt: The Architecture should support XMK and CRD data formats.

SP1_GEN_REQ65_SW_RDE_VOI_v01	This requirement is possible to adapt: The Architecture should be described by UML 2.
SP1_GEN_REQ66_SW_RDE_VOI_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ67_SW_RDE_VOI_v01	It is not clear how to adapt this requirement.

8.1.3 Automotive demonstrator requirements:

SP1_AUT_REQ01_SW_VIS_v01	The requirement is similar to req_3_req_VIT.
SP1_AUT_REQ02_SW_VIS_v01	The requirement is similar to req_4_req_VIT.
SP1_AUT_REQ03_SW_VIS_v01	The requirement is similar to req_5_req_VIT.
SP1_AUT_REQ04_GEN_VIS_v01	The requirement is similar to req_6_req_VIT.
SP1_AUT_REQ05_GEN_VIS_v01	The requirement is similar to req_7_req_VIT.
SP1_AUT_REQ06_SW_VIS_v01	The requirement is similar to req_8_req_VIT.
SP1_AUT_REQ07_SW_VIS_v01	The requirement is similar to req_9_req_VIT.
SP1_AUT_REQ08_SW_VIS_v01	The requirement is similar to req_10_req_VIT.
SP1_AUT_REQ09_SW_VIS_v01	The requirement is similar to req_13_req_VIT.
SP1_AUT_REQ10_SW_VIS_v01	The requirement is similar to req_1_req_VIT.
SP1_AUT_REQ11_UC_VIS_v01	The requirement is similar to req_14_req_VIT.
SP1_AUT_REQ12_UC_VIS_v01	The requirement is similar to req_12_req_VIT.
SP1_AUT_REQ13_HW_VIS_v01	The requirement is similar to req_16_req_VIT.
SP1_AUT_REQ14_HW_VIS_v01	The requirement is similar to req_17_req_VIT.
SP1_AUT_REQ15_SW_VIS_v01	The requirement is similar to req_18_req_VIT.

SP1_AUT_REQ16_SW_VIS_v01	The requirement is similar to req_8_req_VIT.
SP1_AUT_REQ17_PERF_CRF_v01	This requirement is similar to SP1_UAV_REQ01_PERF_SGA_v01. This requirement is possible to adapt: The Architecture shall support large variety of scenarios
SP1_AUT_REQ18_PERF_CRF_v01	This requirement is AUT specific.
SP1_GEN_REQ19_PERF_CRF_v01	This requirement is similar to SP1_MAV_GEN_REQ06_v01. This requirement is possible to adapt: the Architecture shall be able to handle real-time critical situations
SP1_GEN_REQ20_HW_CRF_v01	This requirement is possible to adapt: The Architecture shall support different interfaces and different kinds of networks.
SP1_AUT_REQ21_PERC_CRF_v01	It is not clear how to adapt this requirement.
SP1_AUT_REQ22_PERC_CRF_v01	This requirement is possible to adapt: The Architecture should be able to provide data about weather/environment conditions in an area.
SP1_GEN_REQ23_SW_CRF_v01	This requirement is possible to adapt: The Architecture should work with a coordinate system for positioning objects.
SP1_GEN_REQ24_SW_CRF_v01	This requirement is similar to SP1_MAV_INTFC_REQ21_v01
SP1_GEN_REQ25_SW_CRF_v01	This requirement can be adopted: The Architecture has to ensure that execution time of the modules will not exceed a given threshold.
SP1_AUT_REQ26_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ27_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ28_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific
SP1_AUT_REQ29_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.

SP1_AUT_REQ30_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ31_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ32_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ33_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ34_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ35_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_AUT_REQ36_UC_CRF_v01	It is not clear how to adapt this requirement. This requirement is AUT specific.
SP1_GEN_REQ37_SW_CRF_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ38_SW_CRF_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ39_SW_CRF_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ40_SW_CRF_v01	This requirement is similar to SP1_MAV_TECH_REQ37_v01 This requirement is possible to adapt.
SP1_GEN_REQ41_SW_CRF_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ42_SW_CRF_v01	This requirement is possible to adapt: The Architecture shall standardize interfaces between different components and subsystems.
SP1_GEN_REQ43_SW_CRF_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ44_SW_CRF_v01	It is not clear how to adapt this requirement.
SP1_GEN_REQ45_SW_CRF_v01	This requirement is possible to adapt: The Architecture shall provide confidence level for measured data.

SP1_GEN_REQ46_SW_CRF_v01	This requirement is possible to adapt: The Architecture shall support configurations.
SP1_AUT_REQ47_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ48_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ49_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ50_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ51_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ52_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ53_SW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ54_SW_DLR_v01	This requirement is possible to adapt: The Architecture shall support saving all relevant data for evaluation.
SP1_AUT_REQ55_SW_DLR_v01	This requirement is similar to req_121_eval_DLR
SP1_AUT_REQ56_SW_DLR_v01	This requirement is similar to req_122_eval_DLR
SP1_AUT_REQ57_Gen_DLR_v01	This requirement is similar to req_123_eval_DLR
SP1_AUT_REQ58_SW_DLR_v01	This requirement is possible to adapt: The Architecture has to provide interfaces to integrate external software or hardware elements.
SP1_AUT_REQ59_UC_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ60_SW_DLR_v01	It is not clear what this requirement means.
SP1_AUT_REQ61_HW_DLR_v01	This requirement can be adapted: The Architecture concept has to provide distributed calculation of services/applications. It is similar to the requirement req_125_eval_DLR.
SP1_AUT_REQ62_HW_DLR_v01	It is similar to the requirement req_126_eval_DLR.
SP1_AUT_REQ63_HW_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ64_UC_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ65_UC_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ66_UC_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ67_UC_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ68_UC_DLR_v01	This requirement is AUT specific.
SP1_AUT_REQ69_UC_DLR_v01	This requirement is similar to SP1_MAV_GEN_REQ13_v01.

8.1.4 Maritime demonstrator requirements:

SP1_MAR_REQ01_v01	This requirement is MAR specific.
SP1_MAR_REQ02_v01	This requirement is MAR specific.
SP1_MAR_REQ03_v01	This requirement is MAR specific.
SP1_MAR_REQ04_v01	It is not clear what is meant by this requirement. It is MAR specific, probably.
SP1_MAR_REQ05_v01	This requirement is MAR specific.
SP1_MAR_REQ06_v01	This requirement is MAR specific.
SP1_MAR_REQ07_v01	This requirement is MAR specific.
SP1_MAR_REQ08_v01	This requirement is MAR specific.
SP1_MAR_REQ09_v01	This requirement is MAR specific.
SP1_MAR_REQ10_v01	This requirement is MAR specific.
SP1_MAR_REQ11_v01	This requirement is MAR specific.
SP1_MAR_REQ12_v01	This requirement is MAR specific.
SP1_MAR_REQ13_v01	This requirement is MAR specific.
SP1_MAR_REQ14_v01	This requirement is MAR specific.
SP1_MAR_REQ15_v01	This requirement is MAR specific.
SP1_MAR_REQ16_v01	This requirement is MAR specific.
SP1_MAR_REQ17_v01	This requirement is MAR specific.
SP1_MAR_REQ18_v01	This requirement is MAR specific.
SP1_MAR_REQ19_v01	It is not clear whether employ this requirement for the Architecture. The Architecture is cross-domain but deals with interfaces.
SP1_MAR_REQ20_v01	It is not clear whether employ this requirement for the Architecture. The Architecture is cross-domain but deals with interfaces.

8.2 WP1.2 MTT Requirements

Following are the complete analysis table for the MTT requirements which is already introduced by chapter 4.2.2. A detailed description of the MTT requirements is available by the deliverable [D1-02]:

req_5_req_VIT	It is not clear how the Architecture can support software development tools for embedded systems.
req_6_req_VIT	It is not clear how the Architecture can support standard graphics design tools. We agree that the architecture should be created by using standard modeling tools for architectural design.
req_7_req_VIT	It is not clear how the Architecture can support standard requirements capturing tools.
req_21_dev_LND	It is not clear how the Architecture can keep hardware compatibility.
req_22_dev_LND	It is not clear how the Architecture can support the drag and drop capability.
req_32_dev_LND	It is not clear how the Architecture can provide a graphical development tool.
req_34_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_35_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_36_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_37_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable.
req_38_dev_HON	It is not clear what “the model” is.
req_39_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_40_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_41_dev_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_42_dev_HON	It is not clear what “the model” is.
req_43_dev_HON	It is not clear what “the model” is.
req_44_dev_HON	It is not clear what “the model” is.
req_45_dev_HON	It is not clear what “the model” is.
req_46_dev_HON	It is not clear what “the model” is.
req_47_dev_HON	It is not clear what “the model” is.
req_48_dev_HON	It is not clear what “the model” is.

req_49_dev_HON	It is not clear what “the model” is.
req_50_dev_HON	It is not clear what “the model” is.
req_51_dev_HON	It is not clear what “the model” is.
req_52_dev_HON	It is not clear what “the model” is.
req_54_dev_HON	It is not clear what “the model” is.
req_56_req_HON	Architecture is not an MTT for Requirements Capturing. This requirement is not applicable.
req_57_req_HON	It is not clear how the Architecture can support prioritization of requirements. It seems that requirement capturing tools are out of the scope of the Architecture.
req_58_req_HON	Architecture is not an MTT for Requirements Capturing. This requirement is not applicable.
req_59_req_HON	Architecture is not an MTT for Requirements Capturing. This requirement is not applicable.
req_60_spec_HON	Architecture is not an MTT for Requirements Capturing. This requirement is not applicable.
req_61_spec_HON	Architecture is not an MTT for Requirements Capturing. This requirement is not applicable.
req_62_spec_HON	This requirement seems to be specific to certain MTT and it is not generalizable for the Architecture.
req_63_spec_HON	This requirement is not applicable for the Architecture because it defines structure of textual requirements.
req_74_spec_EAD	It is not clear how the Architecture can support define/crosscheck requirements for DCoS systems.
req_84_spec_EAD	It is not clear how the Architecture could allow to identify inconsistencies in DCoS Systems
req_86_eval_HON	It is not clear what “the model” is.
req_87_eval_HON	It is not clear what “the model” is.
req_88_eval_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_89_eval_HON	This requirement seems to be specific to a certain MTT and it is not generalizable for the Architecture.
req_93_eval_HON	This requirement seems to be specific to a certain MTT. It is not clear how to apply this requirement for the Architecture.
req_94_eval_HON	This requirement seems to be specific to a certain MTT. It is not clear how to apply this requirement for the Architecture.
req_95_eval_HON	This requirement seems to be specific to a certain MTT. It is not clear how to apply this requirement for the Architecture.
req_108_eval_HON	This requirement seems to be specific to a certain MTT. It is not clear how to apply this requirement for the Architecture.
req_110_eval_HON	It is not clear how the Architecture can define criteria for evaluation.
req_115_eval_HON	It is not clear whether it is needed to define different Architectures for different design phases.